Aplikasi Graf dalam Pembuatan Environment Procedural Generation dalam Unity

Muhammad DavisAdhipramana - 13522157[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522157@.std.stei.itb.ac.id

***Abstract*-Environtment Procedural Generation adalah semacam Teknik yang digunakan pada pembuatan video games, yaitu menciptkan dunia atau lingkungan dengan secara automatis. Dalam pembuatannya, terdapat banyak metode yang dapat digunakan seperti, cellular automata, graf, dan perlin noise. Pada makalah ini, akan diimplementasikan struktur data berupa graf sebagai struktur data utamanya dalam mengimplementasikan Environtment Procedural Generation.**

***Keywords*—Graf, Procedural Generation, Perlin Noise**

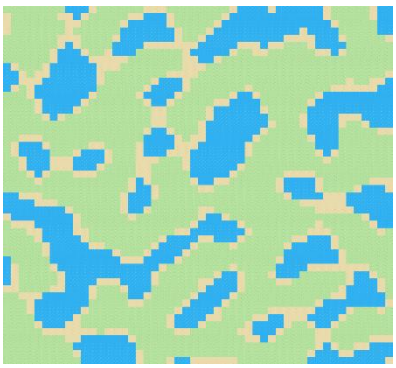
I. PENDAHULUAN

Pembuatan lingkungan atau dunia (environment) dalam video games merupakan fitur penting yang memerlukan perhatian khusus oleh para *game developer*. Proses ini dapat dilakukan secara manual, dengan menempatkan elemen-elemen seperti tile atau sprites secara langsung. Meskipun teknik manual ini sering menghasilkan lingkungan yang detail dan unik, namun dalam pembuatan world yang luas dan banyak fitur yang diulang-ulang, ini kurang praktis karena akan membutuhkan waktu yang cukup lama.

Selain pendekatan manual, dalam pembuatan environtment juga dapat menggunakan metode procedural generation. Ide dari metode ini adalah membangun environment tersebut secara otomatis dengan mengatur aturan tertentu, yang selanjutnya akan terbentuk world baru sesuai aturan yang ditentukan.

Dalam pemilihan metode environtment procedural generation, ada beberapa aspek yang perlu diperhatikan diantaranya dimensi yang digunakan, environment style, camera style, dan boundary dari video games yang ingin dibuat. Dalam Unity, terdapat dua dimensi yang dapat dipilih yaitu 2D dan 3D. Selanjutnya ada beberapa environment style yang dapat diantaranya sandbox, dungeon, space (luar angkasa), dan Rooms. Aspek lainnya adalah camera point of view, camera view untuk 2D dan 3D berbeda diantaranya yaitu first person, third person (3D) dan top-down, isometric (2D), aspek terakhir yang perlu diperhatikan adalah boundary, Batasan ini adalah Batasan untuk environment yang akan dibuat. Contohnya adalah endless dan limited.

Dalam makalah ini, fokus akan diberikan pada implementasi struktur graf berwarna untuk world atau environment procuderal generation pada game berdimensi dua (2D), dengan environment style berupa sandbox, camera berupa top-down, dan boundary berupa limited. Dengan pendekatan ini, akan dihasilkan suatu environment yang berupa environment yang bermacam-macam berdasarkan rules yang ditetapkan pada graf.



Gambar 1 : Contoh Graf Procedural generation

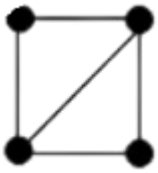
Sumber:
https://www.nikouusitalo.com/content/images/size/w1000/2021/07/ayylmao637165472952097363.png

II. LANDASAN TEORI

A. Graf

Graf adalah suatu struktur data yang memiliki simpul (vertex) dan sisi (edge) sebagai penghubung antar simpul-simpul tersebut. Dalam konteks pembuatan Procedural Generation dengan menggunakan Graph, terdapat beberapa aspek penting yang perlu diperhatikan mengenai Graph:

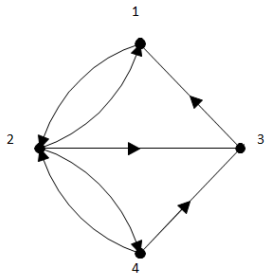
1. **Jenis Graph**: terdapat dua jenis graph, yaitu graph sederhana dan graf tak-sederhana. pada penerapan procedural generation, graph sederhana-lah yang umumnya digunakan. Meskipun demikian, beberapa teknik dalam procedural generation mungkin juga memanfaatkan graf tak-sederhana.



Gambar 2 : Contoh Graf Sederhana

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

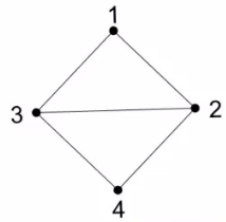
2. **Orientasi Arah Graf** : Terdapat dua jenis orientasi graf, diantaranya graf berarah dan graf tak-berarah. Pada pembuatan procedural generation ini, pada dasarnya graf yang dibuat adalah suatu graf berarah. Hal ini digunakan untuk menghindari node selanjutnya memiliki id yang sama.



Gambar 3 : Contoh Graf Berarah

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

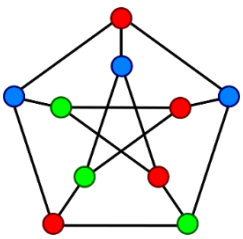
3. **Adjacent (Ketetanggaan)**: Dua buah simpul dikatakan bertetangga jika keduanya terhubung langsung. Konsep ini penting dalam implementasi graf berwarna.



Gambar 4 : Contoh Ketetanggaan sekaligus Graf tak berarah

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf

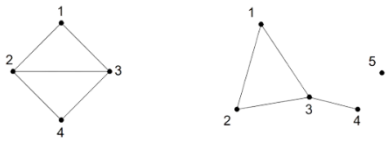
4. **Pewarnaan Graf**: Pewarnaan graf dapat dilakukan pada simpul (node) atau sisi (edge). Dalam konteks procedural generation, penggunaan graf berwarna akan sangat dibutuhkan dalam hubungan antar bioma.



Gambar 5 : Contoh Graf berwarna

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/21-Graf-Bagian3-2023.pdf

5. **Adjacency List**: Merupakan suatu list yang menyimpan matriks-matriks tetangga dari suatu node. Digunakan untuk memastikan bahwa suatu simpul tidak bertetanggaan dengan dirinya sendiri.



Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

(a)

Simpul	Simpul Tetangga
1	2, 3
2	1, 3
3	1, 2, 4
4	3
5	-

(b)

Gambar 6 : Contoh Adjacency List

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/20-Graf-Bagian2-2023.pdf

Masih banyak lagi konsep lain dalam graph yang dapat dijelaskan, namun dalam pembuatan procedural generation ini, lima konsep tersebut lah yang akan digunakan dalam pengimplementasiannya.

Dengan memahami konsep-konsep ini, kita dapat melangkah lebih jauh dalam pemahaman dan implementasi procedural generation dengan pendekatan graph.

B. Seed

Dalam pengimplementasian Procedural Generation, penggunaan angka acak (random number) adalah hal yang umum. Seed merupakan semacam placeholder yang memulai proses menghasilkan angka acak. Sebagai contoh, jika seednya adalah 0 dan nilai acaknya adalah 10, maka setiap kali program dijalankan, nilai acak tersebut akan konsisten.

C. Vector

Pada pengimplementasian Procedural Generation, terutama dalam pengembangan game, vektor (vector) menjadi elemen kunci yang sering digunakan. Beberapa istilah terkait vektor yang relevan meliputi:

1. **Normalisasi Vektor** : Normalisasi vektor adalah proses membuat vektor tersebut memiliki magnitudo (magnitude) sebesar 1. Hal ini berguna, antara lain, untuk menentukan pola acak dalam penggunaan *Perlin Noise* yang akan dibahas nantinya.

2. **Mapping Vektor** : Nilai acak yang didapat berdasarkan Algoritma *Perlin Noise* akan dimapping nantinya pada saat merubah warna elemen didalam node.

D. Procedural generation

Dalam mengimplementasikan Procedural Generation, pemahaman mendalam mengenai teknik ini sangatlah penting. Procedural Generation adalah suatu teknik untuk menciptakan suatu kreasi dengan data dan rules yang ada secara otomatis oleh computer. Teknik ini sering digunakan dalam pembuatan video games atau animasi. Pada makalah ini, fokus utamanya adalah pembuatan world (environtment) dalam video games. Seperti yang sudah dijelaskan pada *Bab Introduction*,

Terdapat berbagai aspek yang perlu diperhatikan, diantaranya:

1 **Dimensi** : Dalam video games terdapat dua space yang dapat digunakan yaitu 2D dan 3D. Pada implementasi ini, akan digunakan 2D space dengan graf sebagai dasar, hal ini mengingat implementasi Procedural Generation pada 3D memerlukan teknik yang lebih kompleks seperti Grammar Graph dan Cellular Automata

Gambar 7 : Contoh Game 2D, Stardew Valley

Sumber:https://www.pcgamer.com/stardew-valley-review/

2 **Environtment Style**: Terdapat berbagai Environment Style yang dapat dipilih, seperti dungeon, sandbox, story, dan rooms. Implementasi kali ini akan berfokus pada gaya Sandbox dimana pada Sandbox elemen yang digunakan bersifat dinamis dan dapat berubah ubah.

Gambar 8 : Contoh Game Sandbox, Minecraft

Sumber: https://www.malavida.com/en/faq/how-to-play-minecraft

3. **Camera Style** : Perlu dipahami bahwa untuk 2D dan 3D space memiliki camera style yang berbeda beda. Pada implementasi ini, akan digunakan top-down camera view dimana camera tersebut akan mengarah diatas playernya.

Gambar 9 : Contoh Game Top-down

4. **Boundary** : Konsep batasan (boundary) menentukan apakah game bersifat infinity atau memiliki batasan tertentu. Pada implementasi ini, ruang lingkup (space) yang digunakan bersifat terbatas.

Selain aspek-aspek tersebut, implementasi Procedural Generation ini akan difokuskan pada struktur graf dengan penggunaan teknik random dan manipulasi vektor.

III. IMPLEMENTASI

A. Graph Stucture

Graph akan menjadi struktur utama yang akan digunakan pada implementasi Procedural Generation ini. Simpul pada graph yang akan dibuat, harus memuat segala informasi yang dibutuhkan dalam pembuatan Procedural generation ini. Dikarenakan banyak impelementasi random yang digunakan, terdapat beberapa informasi yang diperlukan suatu simpul. Informasi tersebut adalah Posisi (Vector), Id, dan Radius.

```
public struct Node
{
    public int BiomeId;
    public Vector2 location;
    public float Radius;
}
```

Gambar 10 : Data simpul

Posisi berfungsi sebagai koordinat lokasi simpul dalam game, sementara Radius menentukan ukuran lingkaran yang merepresentasikan sebuah Benua.BiomeIpada dasarnya memiliki prinsip yang sama dengan pewarnaan simpul. Tiap Id merepresentasikan warna yang berbeda untuk tiap benua,

Selanjutnya akan dibahas mengenai hubungan antar simpul atau sisi. digunakan suatu graf berarah untuk hubungan antar simpul, data yang disimpan adalah data node selanjutnya dan node saat ini berdasarkan urutan kemunculannya dalam graph.

```
public struct Edge
{
    public int FromNodeIndex;
    public int ToNodeIndex;

    1 reference
    public Edge(int from, int to)
    {
        FromNodeIndex = from;
        ToNodeIndex = to;
    }
}
```

Gambar 11 : Data sisi

Selanjutnya adalah struktur data Graph. Class graph menyimpan simpul-simpul dan edge menggunakan metode adjacency list. Dalam penyimpanan data, edge untuk setiap node direpresentasikan dalam suatu list. Hal ini memudahkan akses dan manipulasi hubungan antar simpul.

Selain itu, terdapat beberapa procedure dan fungsi penting saat membuat graf diantaranya penambahan simpul (AddNode) dan edge (AddEdge) dalam list. Serta, Fungsi GetNeighbours yang digunakan untuk mendapatkan daftar simpul yang menjadi tetangga dari suatu node, beserta informasi tentang hubungan yang dimilikinya.

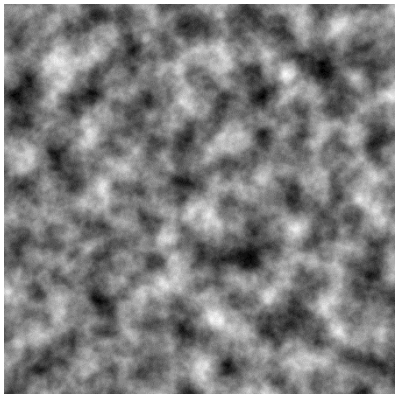
Implementasi ini memastikan bahwa struktur data graph tidak hanya menyimpan informasi secara efisien, tetapi juga menyediakan fungsi-fungsi yang diperlukan untuk memanipulasi dan mengakses data graph dengan mudah.

```
public class WorldGraph
{
    11 references
    public List<Node> Nodes { get; private set; }
    3 references
    public List<Edge> Edges { get; private set; }

    1 reference
    public WorldGraph()
    {
        Nodes = new List<Node>();
        Edges = new List<Edge>();
    }

    1 reference
    public void AddNode(Node node)
    {
        Nodes.Add(node);
    }

    1 reference
    public void AddEdge(int fromNodeIndex, int toNodeIndex)
    {
        Edges.Add(new Edge(fromNodeIndex, toNodeIndex));
    }

    1 reference
    public List<int> GetNeighbors(int nodeIndex)
    {
        List<int> neighbors = new List<int>();

        foreach (Edge edge in Edges)
        {
            if (edge.FromNodeIndex == nodeIndex)
            {
                neighbors.Add(edge.ToNodeIndex);
            }
            else if (edge.ToNodeIndex == nodeIndex)
            {
                neighbors.Add(edge.FromNodeIndex);
            }
        }

        return neighbors;
    }
}
```

Gambar 12 : Struktur Data Graph

B. Perlin Noise

Perlin Noise sendiri adalah suatu generated map yang sangat sering digunakan dalam mengimplementasikan suatu procedural generation.



Gambar 13 : Contoh Perlin Noise

Sumber: https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401

Proses Perlin Noise dimulai dengan pengumpulan data kunci seperti lebar (width), tinggi (height), dan skala (scale). Lebar dan tinggi menentukan ukuran peta noise, sementara skala mengontrol seberapa besar perbesaran yang diterapkan.

```
public class PerlinNoise
{
    public int width = 256;
    public int height = 256;
    public float scale = 20f;
    1 reference
    public PerlinNoise(int width, int height, float scale)
    {
        this.width = width;
        this.height = height;
        this.scale = scale;
    }
```

Gambar 14 : Perlin Noise Data

Setelah data Perlin Noise dikumpulkan, langkah berikutnya adalah menghasilkan Biome Map. Biome Map ini berperan sebagai wadah untuk menyimpan data warna atau tile set, yang akan di-mapping saat bersatu dengan graf.

```
void GenerateBiomeMap()
{
    int width = _perline.width;
    int height = _perline.height;
    float scale = _perline.scale;


    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            float xCoord = (float)x / width * scale;
            float yCoord = (float)y / height * scale;

            float perlinValue = Mathf.PerlinNoise(xCoord, yCoord);

            int colorIdx = _perline.DetermineTiles(perlinValue);

            // Ensure that the indices are integers

            biomeMap[x,y] = colorIdx;
        }
    }
}
```

Gambar 15 : Noise Map data Extraction

Pada setiap iterasi piksel, Perlin Noise di-generate pada posisi tertentu, menghasilkan tingkat intensitas (intensity level) pada piksel tersebut. Intensitas ini akan digunakan nantinya pada fungsi Determine Tiles.

```
public int DetermineTiles(float perlinValue)
{
    if (perlinValue < 0.2)
    {
        return 0;
    }
    else if (perlinValue < 0.35)
    {
        return 1;
    } else if (perlinValue < 0.6)
    {
        return 2;
    }else if(perlinValue < 0.8)
    {
        return 4;
    }
    else
    {
        return 3;
    }
}
```

Gambar 16 : DetemineTiles function

Fungsi Determine Tiles mengembalikan nilai indeks, yang nantinya digunakan untuk mengambil data dari ColorSets. ColorSets adalah koleksi data warna yang merepresentasikan setiap benua. Dalam implementasi ini, terdapat delapan benua yang berbeda, masing-masing direpresentasikan oleh satu warna.

```
public class ColorSets
{

    public Color[] reds, blues, greens, yellows, purples, oranges, cyans, pinks;
    private Color defaultColor = new Color(0.2f, 0.5f, 0.66f);
    1 reference
    public ColorSets()
    {
        reds = new Color[] { new Color(0.8f, 0.2f, 0.2f), new Color(0.7f, 0.3f, 0.3f), new
        blues = new Color[] { new Color(0.2f, 0.2f, 0.8f), new Color(0.3f, 0.3f, 0.7f), ne
        greens = new Color[] { new Color(0.2f, 0.8f, 0.2f), new Color(0.3f, 0.7f, 0.3f), n
        yellows = new Color[] { new Color(0.8f, 0.8f, 0.2f), new Color(0.7f, 0.7f, 0.3f),
        purples = new Color[] { new Color(0.8f, 0.2f, 0.8f), new Color(0.7f, 0.3f, 0.7f),
        oranges = new Color[] { new Color(1f, 0.65f, 0.2f), new Color(0.8f, 0.45f, 0.25f),
        cyans = new Color[] { new Color(0.2f, 0.8f, 0.8f), new Color(0.3f, 0.7f, 0.7f), ne
        pinks = new Color[] { new Color(1f, 0.5f, 0.7f), new Color(0.9f, 0.4f, 0.6f), new
    }

}
```

Gambar 17 : Colorsets Class

Warna ini juga dapat digantikan dengan tiles atau sprite, yaitu elemen gambar yang biasa digunakan dalam pengembangan game sebagai objek visual. Setiap benua memiliki satu warna khas, dan hal ini dapat dilihat pada fungsi determineBiomeColor.

```
public Color[] DetermineBiomeColor(int biomeIndex)
{
    // Use biomeIndex to fetch the corresponding color set from ColorSets
    // Adjust this based on your actual logic for color assignment
    ColorSets _color = new ColorSets();
    switch (biomeIndex)
    {
        case 1:
            return _color.reds; // You might want to randomize this
        case 2:
            return _color.blues; // You might want to randomize this
        case 3:
            return _color.greens; // You might want to randomize this
        case 4:
            return _color.yellows;
        case 5:
            return _color.purples;
        case 6:
            return _color.oranges;
        case 7:
            return _color.cyans;
        case 8:
            return _color.pinks;
    }
    return _color.reds;
}
```

Gambar 18 : Function Determine Biome Color

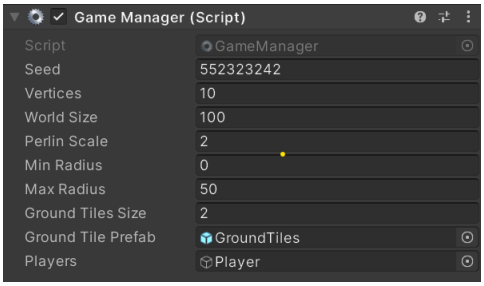
Dengan indeks yang dihasilkan oleh Determine Tiles, hal tersebut dapat digunakan untuk akses ke elemen pada list yang dihasilkan oleh determineBiomeColor. Indeks ini kemudian di-mapping pada biomeMap, sebuah struktur data yang menampung hasil akhir dari proses ini.

Melalui penggunaan Perlin Noise, Procedural Generation dapat menghasilkan variasi dan kompleksitas dalam pembentukan lingkungan game dengan cara yang lebih realistis dan dinamis.

C. Persiapan

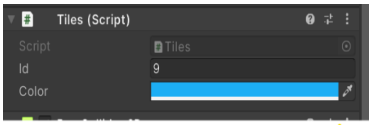
Dalam bagian ini, akan dijelaskan beberapa set-up atau persiapan yang perlu dilakukan sebelum menjalankan program Procedural Generation.

1. **Input Configuration:**
   a. **SeeSeed:**aceholder untuk menyimpan data random
   b. **Vertices**: Menentukan jumlah graf yang akan dibuat, dapat disesuaikan dengan ukuran dunia (world size)
   c. **World Size:** Ukuran dunia yang ditetapkan, dengan bentuk persegi untuk mempertahankan keterbatasan game
   d. **Perlin Size:** Menentukan perbesaran atau pengecilan dari Perlin Noise yang akan di-generate.
   e. **Min Radius dan Max Radius:** Menentukan ukuran minimum dan maksimum untuk radius suatu biome. Nilai ini ditentukan secara acak.
   f. **Ground Tile Size**: Menentukan ukuran tile yang digunakan sebagai ground.
   g. **Players:** Digunakan untuk menentukan posisi awal pemain nantinya.



Gambar 19 : Procedural Data

2. **Ground Tile Generation :**
   Membuat ground tile dengan bentuk persegi memegang dua data penting yaitu Id dan Color.



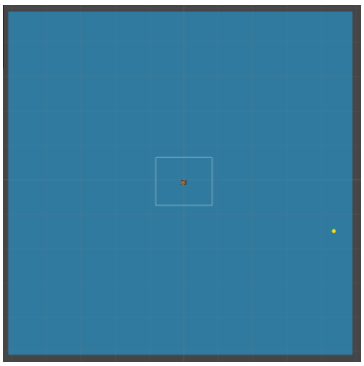
Gambar 20 : GroundTiles Data

Id menggambarkan Id dari Biome yang akan menggantikannya, dengan Id 9 sebagai Id default. Color merupakan data warna yang akan digantikan berdasarkan Id Biome nantinya.

```
private void GenerateGroundTiles()
{
    GameObject groundContainer = new GameObject("GroundTilesContainer");
    float tileSize = groundTilePrefab.transform.localScale.x;
    float _finalWorldSize = worldSize / 2;
    for (float x =  -_finalWorldSize; x < _finalWorldSize; x += tileSize)
    {
        for(float y = -_finalWorldSize; y < _finalWorldSize; y += tileSize)
        {
            InstantiateGroundTile(groundContainer.transform,new Vector2(x, y), 9, new Color(0.2f,0.5f,0.66f));
        }
    }
    groundContainer.transform.SetAsFirstSibling();
}
```

Gambar 21 : Procedure Generate Ground Tiles

Fungsi generateGroundTiles akan menghasilkan ground tile sebanyak (WorldSize * 2) dengan posisi sesuai dengan x dan y dari (-WorldSize / 2) hingga (WorldSize / 2).



Gambar 22 : Hasil Generate Ground Tiles

Set-up ini akan menjadi landasan penting pada tahap pengimplementasian Graph dengan menggunakan Perlin Noise dalam game yang sedang dibuat. Dengan konfigurasi input yang fleksibel, ukuran dunia yang dapat disesuaikan, dan ground tile yang ter-generate dengan baik.

D. Generate Graf

Setelah mengimplementasikan seluruh persiapan diatas, graph sudah siap dibuat. Alur pembuatan dari graf ini berbeda dengan graf biasannya. Dalam pengimplementasian graf dalam procedural generation, untuk tiap simpul yang akan dibentuk, semuanya ditentukan secara random. Berikut Langkah langkah penentuannya:

1. **BiomeID**: Nilai BiomeId dihasilkan secara acak, dengan syarat BiomeId yang dihasilkan tidak sama dengan Id sebelumnya, hal ini diperiksa dengan menggunakan fungsi HasSameBiomeNeighbour.
2. **Radius:** Radius dari biome di-generate secara acak.
3. **Posisi**: saat simpul pertama, pembuatan posisi dibatasi, tetapi selalu berada di ujung world untuk menghindari penempatan biome di tengah, yang dapat menyulitkan pembentukan simpul berikutnya. Pada simpul berikutnya penentuannya akan ditentukan berdasarkan simpul sebelumnya.
4. **Random Neighbour Index:** digunakan untuk menciptakan konektivitas yang beragam, tetapi, tidak boleh dengan dirinya sendiri.

```
1 reference
private void GenerateWorldGraph()
{
    for(int i = 0; i < vertices; i++)
    {
        int biomeId;

        biomeId = Random.Range(1, 8);
        float angle, distance;
        float nodeRadius = Random.Range(minRadius, maxRadius);
        if (WorldGraph.Nodes.Count == 0)
        {
            angle = Random.Range(0f, 360f) * Mathf.Deg2Rad;
            worldSize /= 2;
            distance = Random.Range(-worldSize,0.5f * -worldSize);
        }
        else
        {
            angle = Random.Range(0f, 360f) * Mathf.Deg2Rad;
            distance = Random.Range(WorldGraph.Nodes[i - 1].Radius, 3 * WorldGraph.Nodes[i - 1].Radius);
            while (distance < (0.3f * WorldGraph.Nodes[i - 1].Radius))
            {
                distance *= 1.5f; // Adjust the multiplier for subsequent nodes
            }
        }

        float x = distance * Mathf.Cos(angle);
        float y = distance * Mathf.Sin(angle);

        Vector2 nodePosition = new Vector2(x, y);
        Node newNode = new () { BiomeId= biomeId, location = nodePosition , Radius = nodeRadius};
        WorldGraph.AddNode(newNode);
        if (WorldGraph.Nodes.Count > 1)
        {
            int randomNeighborIndex;
            do
            {
                randomNeighborIndex = Random.Range(0, WorldGraph.Nodes.Count);
            } while (randomNeighborIndex == i || WorldGraph.Nodes[randomNeighborIndex].BiomeId == biomeId);
            WorldGraph.AddEdge(i, randomNeighborIndex);
        }

    }
}
```

Gambar 23 : Procedure untuk menghasilkan Graf berdasarkan aturan graf berwarna, serta dilakukan secara random

E. Apply Tiles

Setelah Graf dibuat, Langkah terakhir adalah menerapkan rules yang sudah dibuat dari campuran graf dan perlin noise pada groundTiles sebagai placeholdernya. Berikut Langkah penerapannya :

1. **Buat Pseudo Collider :** suatu collider dibuat dengan bentuk lingkaran dengan posisi simpul sebagai pusatnya dan radius simpul untuk radiusnya. Collider ini sendiri berfungsi untuk mendeteksi adanya collision atau tabrakan dengan collider lain.
2. **Identifikasi Tiles** : Untuk tiap collision yang terdeteksi dalam Pseudo Collider, id tile diambil.
3. **Pemrosesan Collision** : Sebelumnya, diambil terlebih dahulu warna yang berseuaian dengan idBiome saat ini dengan fungsi DetermineBiomeColor. Pada program yang dibuat, BiomeId yang dimaksud adalah newId. Kemudian akan dicek index yang dihasilkan pada mapping biomeMap. Jika index tersebut bukan 4 (warna default), box collision dinonaktifkan untuk menghindari pengaruh collision pada pemain. Index 4 dipilih sebagai warna default karena untuk mencegah tersbentuk seluruh benua berbentuk lingkaran penuh yang terasa kurang natural.
4. **Ubah Warna**: digunakan fungsi ChangeColor untuk merubah warna dari groundTile

```
private void UpdateGroundTilesInRadius(Vector2 center, float radius, int newId)
{
    Collider2D[] hitColliders = Physics2D.OverlapCircleAll(new Vector2(center.x, center.y), radius);
    //Collider2D[] hitColliders = Physics2D.OverlapBoxAll(new Vector2(center.x, center.y), new Vector2(radius
    bool isPlayerPlaced = false;
    //int colorIdx = _perline.biomeMap[x, y];
    foreach (var hitCollider in hitColliders)
    {
        Tiles groundTile = hitCollider.GetComponent<Tiles>();
        if (groundTile != null)
        {
            if(groundTile.Id == 9)
            {
                groundTile.Id = newId;
                Color[] temp = _perline.DetermineBiomeColor(newId);
                int xIndex = Mathf.FloorToInt(groundTile.transform.position.x + (worldSize));
                int yIndex = Mathf.FloorToInt(groundTile.transform.position.y + (worldSize));
                int ColorId = biomeMap[(int)xIndex, (int)yIndex];
                groundTile.Color = temp[ColorId];
                if(ColorId != 4)
                {
                    if (!isPlayerPlaced)
                    {
                        Vector3 playerPos = new(groundTile.transform.position.x,groundTile.transform.position
                        Players.transform.position = playerPos;
                        isPlayerPlaced = true;
                    }
                    groundTile.GetComponent<BoxCollider2D>().enabled = false;
                }

                groundTile.ChangeColor();
                // Optionally, update other properties based on the biome ID
            }
        }
    }
```

Gambar 24 : Procedure untuk update data Ground Tiles berdasarkan node biomes saat itu

Setelah semua langkah sudah dilakukan, Langkah yang tersisa adalah meletakkan seluruh fungsi yang ada pada procedure Start. pada unity, procedure start adalah suatu procedure yang akan dijalankan diawal Ketika game di run.

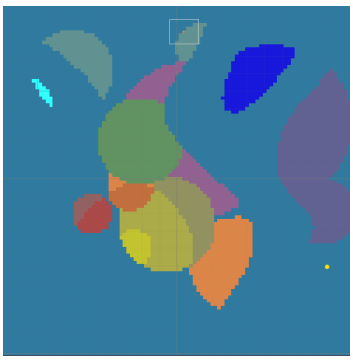
```
private void Start()
{
    groundTilePrefab.transform.localScale = new Vector3(groundTilesSize,groundTilesSize,1);
    biomeMap = new int[worldSize, worldSize];
    Random.InitState(seed);
    WorldGraph = new WorldGraph();
    _perline = new PerlinNoise(worldSize, worldSize, perlinScale);
    GenerateGroundTiles();
    GenerateWorldGraph();
    UpdateGroundTilesFromGraph();
}
```

Gambar 25 : Procedure untuk memulai program pada Unity

IV. HASIL IMPLEMENTASI

Procedural generation yang telah dibuat dapat membuat berbagai macam hasil dengan modifikasi yang beragam. Berikut merupakan beberapa hasil percobaan yang dilakukan:

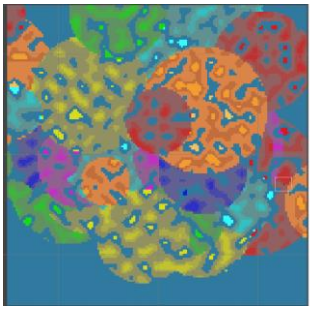
1. **Percobaan 1, dengan seed 232**
   - Jumlah Vertices: 10
   - Ukuran Dunia: 200
   - Skala Perlin Noise: 2
   - Radius Minimum: 10
   - Radius Maksimum: 50
   - Ukuran Ground Tiles: 2



Gambar 26 : Hasil Percobaan 1.

Kombinasi ini menghasilkan sebuah dunia yang terdiri dari 10 bioma(vertices) dengan ukuran dan karakteristik yang berbeda beda. Graf yang terbentuk menjelaskan hubungan antara graf berwarna dimana tidak ada satupun benua dengan warna yang sama berada di sebelah satu sama lain. Berbagai corak yang dihasilkan merupakan hasil yang didapat dari perlin noise dengan skala 2.

2. **Percobaan 2, dengan seed 2324212**
   - Jumlah Vertices: 40
   - Ukuran Dunia: 300
   - Skala Perlin Noise: 20
   - Radius Minimum: 20,
   - Radius Maksimum: 80
   - Ukuran Ground Tiles: 2



Gambar 27 : Hasil Percobaan 2.

Kombinasi ini menghasilkan dunia yang lebih kompleks dan penuh variasi. Dengan meningkatkan jumlah vertices menjadi 40, graf membentuk lebih banyak hubungan antar-benua, menciptakan lanskap yang lebih kompleks dan terhubung dengan baik. Perubahan pada skala perlin noise (perlinScale) memberikan variasi yang lebih bercorak pada benua, semakin besar skala-nya, noise yang dihasilkan akan semakin bercorak.

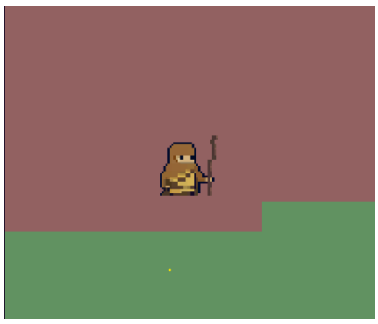
3. **Percobaan 3, dengan seed 232**
   - Jumlah Vertices : 20
   - Ukuran Dunia : 300
   - Skala Perlin Noise : 0.2
   - Radius Minimum : 40
   - Radius Maximum : 100
   - Ukuran Ground Tiles : 1.

Gambar 28 : Hasil Percobaan 3.

Kombinasi ini menghasilkan graf dengan simpul yang sangat berdekatan, hal ini dikarenakan besarnya radius yang dihasilkan, terlalu besar untuk world dengan size yang hanya 300. Dengan perlinscale yang sangat kecil juga menghasilkan corak yang sangat tidak natural. Secara graf, terdapat pelanggaran yaitu terdapat 2 bioma bersebelahan dengan warna yang sama, hal ini disebabkan karena simpul yang terlalu dekat menyebabkan node tetangga dari bioma dengan warna oranye dibawah memiliki idBioma untuk warna biru, begitu juga dengan bioma dengan warna ungu, memiliki idBioma yang sama. Hal ini secara visual melanggar aturan, namun jika digambarkan dalam graf, kedua benua tersebut tidak berhubungan.

Berdasarkan ketiga eksperimen tersebut, dapat ditunjukkan bahwa dengan memodifikasi parameter seperti seed, jumlah vertices, ukuran dunia, dan skala perlin, kita dapat menghasilkan berbagai kombinasi lingkungan yang unik.

Selanjutnya, perlu dicatat bahwa semua gambar yang dihasilkan memperlihatkan perspektif wide view untuk memberikan gambaran dunia dari jauh. Pada kenyataannya, dalam perspektif pemain, hanya sedikit saja yang terlihat.



Gambar 28 : Sudut pandang player

V. KESIMPULAN

Penggunaan graf dalam procedural generation merupakan pilihan yang tepat karena representasi hubungan antar elemen dengan adjacency list memungkinkan pengambilan akses yang lebih simple dan efisien. Graf berwarna juga memberikan identitas unik pada setiap bioma pada *environment* yang dibuat, memberikan keberagaman dan visualisasi yang jelas. Penggabungan dengan metode random dan perlin noise juga merupakan elemen kunci dalam pendistribusian elemen warna pada tiap biomes. Dengan demikian, penggabungan Graph, randomization, dan perlin noise menciptakan suatu environment yang dapat dibuat secara otomatis dan menjadi unik jika value nya diubah. Keseluruhan, makalah ini memberikan wawasan dalam penggunaan teknik graf dalam pengaplikasiannya pada

procuderal generation.

DAFTAR PUSTAKA

[1] B.V. Jessica, *Procuderal Generation: Creating 3D Worlds With Deep Learning.* https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html, diakses pada 9 December 2023.

[2] U. Nico, *Generating a Procedural 2D Map in c#: Part1 – The Attempt.* https://www.nikouusitalo.com/blog/attempting-to-generate-a-procedural-2d-map-in-c-part-1/, diakses pada 9 December 2023.

[3] Munir, Rinaldi, *Graf Bag.1*, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 9 December 2023.

[4] Munir, Rinaldi, *Graf Bag.3*, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/21-Graf-Bagian3-2023.pdf , diakses pada 9 December 2023.

[5] Priabada, Danang, *Random Generaotor : Apa itu seed?. https://callmedanbo.medium.com/random-generator-apa-itu-seed-338fa1228841#:~:text=Seed%20adalah%20nilai%20awal%20yang,acak%20untuk%20banyak%20aplikasi%20praktis, diakses pada 10 December 2023*

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jatinangor, 10 Desember 2023

Muhammad Davis Adhipramana 13522157